CSE5P: Assignment 4 (Due: Wednesday February 13, 11:59PM)

TA: Trung Nguyen, Instructor: Peter Alvaro

Spring 2019

Instructions

Please read the following requirements carefully before coding. You will be penalized **15 points** if you fail to comply **any** of them:

1. Code must be written in Python3 (NOT Python2)
2. You are asked to download the `hw4.py` file on Canvas and modify it
3. You are only allowed to modify some specific portions of the provided `hw4.py` python files. **DO NOT** modify any existing line of the provided skeleton code. Where you can code and where you cannot is already noted by comments in the python files
4. **DO NOT** put the `hw4.py` in a folder or zip it, also **DO NOT** change the file name! You must remain its name as `hw4.py` and submit it to Canvas once you are done
5. There will be no test script for this assignment

Question 1. The Maze Runner [40 pts]

A robot is put in a maze where it can only take a single step of length 1 at a time. The step must be in one of the four following directions: up, down, left, or right. Based on the structure of the maze, a combination of steps in form of a string is provided to the robot so that it can find the right way to go through the maze. The string consists of a combination of directions of each step the robot has to take. That is:

- A character 'u' or 'U' indicates taking a step up
- A character 'd' or 'D' indicates taking a step down
- A character 'l' or 'L' indicates taking a step to the left
- A character 'r' or 'R' indicates taking a step to the right

For example, an input string 'RuLUrd' indicates that the robot must first take a step to the right, then a step up, then a step to the left, then a step up, then a step to the right and finally a step down in order to exit the maze.

1. Modify the function `step_count()` in file `hw4.py` so that this function takes the instruction string as input and returns the number of steps the robot has to take in order to go through the maze.

   Example:

```
>>> step_count('LRlURRDu')
8
```

2. Modify the function `distance()` in file `hw4.py` so that this function takes the instruction string as input and returns the distance (rounded to four decimal digits) between where the robot starts and the exit of the maze.

   Example:

```
>>> distance('LRlURRDu')
1.4142
```

You can assume that the inputs for both the function are always a string. However, if that input string consists of an invalid character, which is a character not in the set of {'r','R','l','L','u','U', 'd', 'D'}, both of the above functions must return -1

Example:

```
>>> step_count('7LRlURRDu')
-1
>>> distance('aLRlURRDu')
-1
```

For distance calculation, we accept error within the range [-0.0001, +0.0001] so you don't have to worry about the error in rounding up or down

Question 2. Data Compression [30 pts]

Most of us have some archiver utility software like Winrar or 7-Zip installed on our personal computers. They help combine multiple files/folders into a single file, which is more convenient to send, receive and manage. But did you notice that the zip or rar files produced by these software are usually smaller in size than the original files/folders, and wonder why? It turns out that these software use several Data Compression algorithms to reduce the file size, while still maintaining all of the information that the file contains. This is very useful, because not only it reduces the storage needed to store the file, but also the bandwidth required to transmit it.

In this problem, you are asked to implement a simple data compression algorithm which is very effective against strings with repetitive characters. This algorithm replaces every sub-string containing multiple identical consecutive characters with a copy of the character and a count of how many times it is repeated. For example, the string 'AAAAATTTTGGGGCCCCAAA', through the algorithm, is compressed to 'A5T4G4C4A3', which can now be interpreted as a sequence of five As, four Ts, four Gs, four Cs, and three As. Notice that the length of the output string is 10 which is only half of the input string's. In this case, we only need half the space to store the string, and half the internet bandwidth to send it.

Modify the function `compress()` in file `hw4.py` so that this function takes a string as input and returns the compressed string.

Example:

```
>>> compress('')
''
>>> compress('a')
'a1'
>>> compress('AAAAATTTTGGGGCCCCAAA')
'A5T4G4C4A3'
>>> compress('##########')
'#10'
>>> compress('zzz ZZ  zZzZ')
'z3 1Z2 2z1Z1z1Z1'
```

For simplicity, you can assume that the input for this function is always a string which does not contain any numeric character.

Question 3. Feeding Frenzy [30 pts + 20 pts extra credit]

Your fish Frenzy is put in a 2D rectangular tank represented by a 2D list containing integers indicating the levels of other fish in the tank. For example, the 2D list `[[1,1,1,1],[2,2,2,2]]` represents a tank of size 4 × 2, with four level 1 fish in the first row and four level 2 fish in the second row. Frenzy can eat a fish if its level is smaller than Frenzy's, and will gain a number of experience equal to the level of the fish it eats. Frenzy will be eaten if the other fish's level is larger or equal to its level.

1. You want to check if there is any fish in the input tank which Frenzy can eat.

   Modify the function `any_smaller()` in `hw4.py` so that it takes the tank (2D list) as the first input, the level of Frenzy (`int` type) as the second input. It returns `True` if there is **any** fish in the tank which Frenzy can eat, and return `False` otherwise.

   Example:

   ```
   >>> any_smaller([[5,4,3],
                    [6,7,8],
                    [4,7,8]], 3)
   False
   >>> any_smaller([[5,4,3],
                    [6,7,8],
                    [4,7,8]], 4)
   True
   ```

2. You want to check if Frenzy can eat all the fish in the input tank.

   Modify the function `all_smaller()` in `hw4.py` so that it takes the tank (2D list) as the first input, the level of Frenzy (`int` type) as the second input. It returns `True` if Frenzy can eat **all** other fish in the tank, and return `False` otherwise.

   Example:

```
>>> all_smaller([[5,4,3],
                 [6,7,8],
                 [4,7,8]], 8)
False
>>> all_smaller([[5,4,3],
                 [6,7,8],
                 [4,7,8]], 10)
True
```

3. (20 pts extra credit) Assume that the search range of Frenzy is a 2 × 2 window. That is, Frenzy can eat maximum 4 fish in any square of size 2 × 2 of the tank, and gain experience points equal to the sum of the levels of the fish in the 2 × 2 square range. However, Frenzy will be eaten if any of the four fish's level is larger or equal to its level. You want Frenzy to gain as much experience as possible, but also avoid being eaten.

   Modify the function `max_exp()` in `hw4.py` so that it takes the tank (2D list) as the first input, the level of Frenzy (`int` type) as the second input. It returns the maximum experience Frenzy can gain in that tank without being eaten.

   For example, in the tank below, Frenzy is a level greater than all the fish in the tank so by eating the fish in square [[7,8],[7,8]], he can maximize his experience points, which is 30.

```
>>> max_exp([[5,4,3],
             [6,7,8],
             [4,7,8]], 9)
30
```

   In the other tank below, the square [[7,7],[6,5]] contains the maximum experience points, but because Frenzy's level is only 7 he can't choose this square and has to pick the [[5,5],[5,5]] square instead.

```
>>> max_exp([[7,7,7,7,7],
             [1,6,5,5,5],
             [9,7,5,5,1]], 7)
20
```

You can assume that the first input for this function is always a 2D list (the tank), and the second input is always an `int` (Frenzy's level), so you don't have to worry about invalid input.